1

AD-A285 108

NTATION PAGE

Form Approved	
OPM No.	

ated to average 1 hour per response, including the time for reviewing instructions, searching existing data wing the collection of information. Send comments regarding this burden, to Washington Headquarters 215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Information and on, DC 20503.

1. AGENCY USE (Leave	2. REPORT	3. REPORT TYPE AND DATES

4. TITLE AND: Compiler: SKI Computers, Inc.
Compiler: SKYvec ADA, Release 3.6
Host: SPARCstation 10 Model 402 (under SunOS 4.1.3)
Target: SKYbolt Model 8146-V (under SKYmpxrt release 3.6)

5. FUNDING

6. AUTHORS:

Wright-Patterson AFB, Dayton, OH

7. PERFORMING ORGANIZATION NAME (S) AND
Ada Validating Facility, Language Control Facility ASB/SCEL, Building 676, Rm. 135
Wright-Patterson AFB, Dayton, OH 45433

8. PERFORMING ORGANIZATION

9. SPONSORING/MONITORING AGENCY NAME(S) AND
Ada Joint Program Office, Defense Information System Agency
Code TXEA, 701 S. Courthouse Rd., Arlington, VA
22204-2199

10. SPONSORING/MONITORING AGENCY

11. SUPPLEMENTARY

12a. DISTRIBUTION/AVAILABILITY: Approved for public release; distribution unlimited

12b. DRISTRIBUTION

13. *(Maximum 200*

SKY Computers, Inc., 940803W1.11374

14. SUBJECT: Ada Programming Language, Ada Compiler Validation Summary Report, Ada Compiler Val. Capability Val. Testing, Ada Val. Office, Ada Val. Facility ANSI/Mil-STD-1815A

15. NUMBER OF

16. PRICE

17 SECURITY CLASSIFICATION	18. SECURITY	19. SECURITY CLASSIFICATION	20. LIMITATION OF
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED

NSN

AVF Control Number: AVF-VSR-602.0894
Date VSR Completed: 8 August 1994
94-06-22-SKY

Ada COMPILER VALIDATION SUMMARY REPORT:

Certificate Number: 940803W1.11374
SKY Computers, Inc.
SKYvec ADA, Release 3.6
SPARCstation 10, Model 402 under SunOS, 4.1.3 =>
SKYbolt Model 8146-V under SKYmpxrt, release 3.6

(Final)

Prepared By:
Ada Validation Facility
645 CCSG/SCSL
Wright-Patterson AFB OH 45433-5707

Accesion For		
NTIS CRA&I		☒
DTIC TAB		☐
Unannounced		☐
Justification		
By		
Distribution /		
Availability Codes		
Dist	Avail and / or Special	
A-1		

4298

Certificate Information

The following Ada implementation was tested and determined to pass ACVC 1.11. Testing was completed on 3 August 1994.

Compiler Name and Version:	SKYvec ADA, Release 3.6
Host Computer System:	SPARCstation 10, Model 402 under SunOS, 4.1.3
Target Computer System:	SKYbolt Model 8146-V under SKYmpxrt, release 3.6
Customer Agreement Number:	94-06-22-SKY

See section 3.1 for any additional information about the testing environment.

As a result of this validation effort, Validation Certificate 940803W1.11374 is awarded to SKY Computers, Inc. This certificate expires two years after MIL-STD-1815B is approved by ANSI.

This report has been reviewed and is approved.

Dale E Lange

Ada Validation Facility
Dale E. Lange
Technical Director
645 CCSG/SCSL
Wright-Patterson AFB OH 45433-5707

[illegible]

Ada Validation Organization
Director, Computer and Software Engineering Division
Institute for Defense Analyses
Alexandria VA 22311

[illegible]

Ada Joint Program Office
Donald J. Reifer
Director, AJPO
Defense Information Systems Agency,
Center for Information Management

DECLARATION OF CONFORMANCE

Customer: *Sky Computers, Inc.*

Ada Validation Facility: CTA Inc.
5100 Springfield Pike, Suite 100
Dayton, Ohio 45431

ACVC Version: 1.11

Ada Implementation:

Compiler Name and Version: *SKYvec ADA release 3.6*

Host Computer System: *SPARCstation-10, Model 402, Sun Microsystems*
Host Operating System: *SunOS 4.1.3*

Target Computer System: *SKYbolt Model 8146-V*
Target Operating System: *SKYmpxrt release 3.6*

Customer's Declaration

I, the undersigned, representing SKY Computers, Inc., declare that SKY Computers, Inc. has no knowledge of deliberate deviations from the Ada Language Standard ANSI/MIL-STD-1815A in the implementation listed in this declaration.
I declare that SKY Computers is the owner of the above implementation and the certificates shall be awarded in the name of the owner's corporate name.

Leo Mirkin Date: July 1, 1994

Leo Mirkin
Manager, Languages & Tools,
SKY Computers, Inc.
27 Industrial Ave.
Chelmsford, MA 01824

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The Ada implementation described above was tested according to the Ada Validation Procedures [Pro92] against the Ada Standard [Ada83] using the current Ada Compiler Validation Capability (ACVC). This Validation Summary Report (VSR) gives an account of the testing of this Ada implementation. For any technical terms used in this report, the reader is referred to [Pro92]. A detailed description of the ACVC may be found in the current ACVC User's Guide [UG89].

1.1 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the Ada Certification Body may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject implementation has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from the AVF which performed this validation or from:

National Technical Information Service
5285 Port Royal Road
Springfield VA 22161

Questions regarding this report or the validation test results should be directed to the AVF which performed this validation or to:

Ada Validation Organization
Computer and Software Engineering Division
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311-1772

1.2 REFERENCES

[Ada83] Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

[Pro92] Ada Compiler Validation Procedures, Version 3.1, Ada Joint Program Office, August 1992.

[UG89] Ada Compiler Validation Capability User's Guide, 21 June 1989.

1.3 ACVC TEST CLASSES

Compliance of Ada implementations is tested by means of the ACVC. The ACVC contains a collection of test programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable. Class B and class L tests are expected to produce errors at compile time and link time, respectively.

The executable tests are written in a self-checking manner and produce a PASSED, FAILED, or NOT APPLICABLE message indicating the result when they are executed. Three Ada library units, the packages REPORT and SPPRT13, and the procedure CHECK_FILE are used for this purpose. The package REPORT also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The package SPPRT13 is used by many tests for Chapter 13 of the Ada Standard. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for Chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. If these units are not operating correctly, validation testing is discontinued.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that all violations of the Ada Standard are detected. Some of the class B tests contain legal Ada code which must not be flagged illegal by the compiler. This behavior is also verified.

Class L tests check that an Ada implementation correctly detects violation of the Ada Standard involving multiple, separately compiled units. Errors are expected at link time, and execution is attempted.

In some tests of the ACVC, certain macro strings have to be replaced by implementation-specific values — for example, the largest integer. A list of the values used for this implementation is provided in Appendix A. In addition to these anticipated test modifications, additional changes may be required to remove unforeseen conflicts between the tests and implementation-dependent characteristics. The modifications required for this implementation are described in section 2.3.

For each Ada implementation, a customized test suite is produced by the AVF. This customization consists of making the modifications described in the preceding paragraph, removing withdrawn tests (see section 2.1), and possibly removing some inapplicable tests (see section 2.2 and [UG89]).

In order to pass an ACVC an Ada implementation must process each test of the customized test suite according to the Ada Standard.

1.4 DEFINITION OF TERMS

Ada Compiler	The software and any needed hardware that have to be added to a given host and target computer system to allow transformation of Ada programs into executable form and execution thereof.
Ada Compiler Validation Capability (ACVC)	The means for testing compliance of Ada implementations, consisting of the test suite, the support programs, the ACVC user's guide and the template for the validation summary report.
Ada Implementation	An Ada compiler with its host computer system and its target computer system.
Ada Joint Program Office (AJPO)	The part of the certification body which provides policy and guidance for the Ada certification system.
Ada Validation Facility (AVF)	The part of the certification body which carries out the procedures required to establish the compliance of an Ada implementation.
Ada Validation Organization (AVO)	The part of the certification body that provides technical guidance for operations of the Ada certification system.
Compliance of an Ada Implementation	The ability of the implementation to pass an ACVC version.
Computer System	A functional unit, consisting of one or more computers and associated software, that uses common storage for all or part of a program and also for all or part of the data necessary for the execution of the program; executes user-written or user-designated programs; performs user-designated data manipulation, including arithmetic operations and logic operations; and that can execute programs that modify themselves during execution. A computer system may be a stand-alone unit or may consist of several inter-connected units.

Conformity	Fulfillment by a product, process, or service of all requirements specified.
Customer	An individual or corporate entity who enters into an agreement with an AVF which specifies the terms and conditions for AVF services (of any kind) to be performed.
Declaration of Conformance	A formal statement from a customer assuring that conformity is realized or attainable on the Ada implementation for which validation status is realized.
Host Computer System	A computer system where Ada source programs are transformed into executable form.
Inapplicable test	A test that contains one or more test objectives found to be irrelevant for the given Ada implementation.
ISO	International Organization for Standardization.
LRM	The Ada standard, or Language Reference Manual, published as ANSI/MIL-STD-1815A-1983 and ISO 8652-1987. Citations from the LRM take the form "<section>.<subsection>:<paragraph>."
Operating System	Software that controls the execution of programs and that provides services such as resource allocation, scheduling, input/output control, and data management. Usually, operating systems are predominantly software, but partial or complete hardware implementations are possible.
Target Computer System	A computer system where the executable form of Ada programs are executed.
Validated Ada Compiler	The compiler of a validated Ada implementation.
Validated Ada Implementation	An Ada implementation that has been validated successfully either by AVF testing or by registration [Pro92].
Validation	The process of checking the conformity of an Ada compiler to the Ada programming language and of issuing a certificate for this implementation.
Withdrawn test	A test found to be incorrect and not used in conformity testing. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains erroneous or illegal use of the Ada programming language.

CHAPTER 2

IMPLEMENTATION DEPENDENCIES

2.1 WITHDRAWN TESTS

The following tests have been withdrawn by the AVO. The rationale for withdrawing each test is available from either the AVO or the AVF. The publication date for this list of withdrawn tests is 22 November 1993.

B27005A	E28005C	B28006C	C32203A	C34006D	C35507K
C35507L	C35507N	C35507O	C35507P	C35508I	C35508J
C35508M	C35508N	C35702A	C35702B	C37310A	B41308B
C43004A	C45114A	C45346A	C45612A	C45612B	C45612C
C45651A	C46022A	B49008A	B49008B	A54B02A	C55B06A
A74006A	C74308A	B83022B	B83022H	B83025B	B83025D
C83026A	B83026B	C83041A	B85001L	C86001F	C94021A
C97116A	C98003B	BA2011A	CB7001A	CB7001B	CB7004A
CC1223A	BC1226A	CC1226B	BC3009B	BD1B02B	BD1B06A
AD1B08A	BD2A02A	CD2A21E	CD2A23E	CD2A32A	CD2A41A
CD2A41E	CD2A87A	CD2B15C	BD3006A	BD4008A	CD4022A
CD4022D	CD4024B	CD4024C	CD4024D	CD4031A	CD4051D
CD5111A	CD7004C	ED7005D	CD7005E	AD7006A	CD7006E
AD7201A	AD7201E	CD7204B	AD7206A	BD8002A	BD8004C
CD9005A	CD9005B	CDA201E	CE2107I	CE2117A	CE2117B
CE2119B	CE2205B	CE2405A	CE3111C	CE3116A	CE3118A
CE3411B	CE3412B	CE3607B	CE3607C	CE3607D	CE3812A
CE3814A	CE3902B				

2.2 INAPPLICABLE TESTS

A test is inapplicable if it contains test objectives which are irrelevant for a given Ada implementation. Reasons for a test's inapplicability may be supported by documents issued by the ISO and the AJPO known as Ada Commentaries and commonly referenced in the format AI-ddddd. For this implementation, the following tests were determined to be inapplicable for the reasons indicated; references to Ada Commentaries are included as appropriate.

The following 201 tests have floating-point type declarations requiring more digits than SYSTEM.MAX_DIGITS:

C24113L..Y (14 tests)	C35705L..Y (14 tests)
C35706L..Y (14 tests)	C35707L..Y (14 tests)
C35708L..Y (14 tests)	C35802L..Z (15 tests)
C45241L..Y (14 tests)	C45321L..Y (14 tests)
C45421L..Y (14 tests)	C45521L..Z (15 tests)
C45524L..Z (15 tests)	C45621L..Z (15 tests)
C45641L..Y (14 tests)	C46012L..Z (15 tests)

C35713B, C45423B, B86001T, and C86006H check for the predefined type SHORT_FLOAT; for this implementation, there is no such type.

C35713D and B86001Z check for a predefined floating-point type with a name other than FLOAT, LONG_FLOAT, or SHORT_FLOAT; for this implementation, there is no such type.

C45423A, C45523A, and C45622A check that the proper exception is raised if MACHINE_OVERFLOWS is TRUE and the results of various floating-point operations lie outside the range of the base type; for this implementation, MACHINE_OVERFLOWS is FALSE.

C45531M..P and C45532M..P (8 tests) check fixed-point operations for types that require a SYSTEM.MAX_MANTISSA of 47 or greater; for this implementation, MAX_MANTISSA is less than 47.

B86001Y uses the name of a predefined fixed-point type other than type DURATION; for this implementation, there is no such type.

CA2009C and CA2009F check whether a generic unit can be instantiated before its body (and any of its subunits) is compiled; this implementation creates a dependence on generic units as allowed by AI-00408 and AI-00506 such that the compilation of the generic unit bodies makes the instantiating units obsolete. (See section 2.3.)

LA3004A..B, EA3004C..D, and CA3004E..F (6 tests) check pragma INLINE for procedures and functions; this implementation does not support pragma INLINE.

CD1009C checks whether a length clause can specify a non-default size for a floating-point type; this implementation does not support such sizes.

CD2A84A, CD2A84E, CD2A84I..J (2 tests), and CD2A84O use length clauses to specify non-default sizes for access types; this implementation does not support such sizes.

BD8001A, BD8003A, BD8004A..B (2 tests), and AD8011A use machine code insertions; this implementation provides no package MACHINE_CODE.

AE2101C and EE2201D..E (2 tests) use instantiations of package SEQUENTIAL_IO with unconstrained array types and record types with discriminants without defaults; these instantiations are rejected by this compiler.

AE2101H, EE2401D, and EE2401G use instantiations of package DIRECT_IO with unconstrained array types and record types with discriminants without defaults; these instantiations are rejected by this compiler.

The tests listed in the following table check that USE_ERROR is raised if the given file operations are not supported for the given combination of mode and access method; this implementation supports these operations.

Test	File Operation	Mode	File Access Method
CE2102D	CREATE	IN_FILE	SEQUENTIAL_IO
CE2102E	CREATE	OUT_FILE	SEQUENTIAL_IO
CE2102F	CREATE	INOUT_FILE	DIRECT_IO
CE2102I	CREATE	IN_FILE	DIRECT_IO
CE2102J	CREATE	OUT_FILE	DIRECT_IO
CE2102N	OPEN	IN_FILE	SEQUENTIAL_IO
CE2102O	RESET	IN_FILE	SEQUENTIAL_IO
CE2102P	OPEN	OUT_FILE	SEQUENTIAL_IO
CE2102Q	RESET	OUT_FILE	SEQUENTIAL_IO
CE2102R	OPEN	INOUT_FILE	DIRECT_IO
CE2102S	RESET	INOUT_FILE	DIRECT_IO
CE2102T	OPEN	IN_FILE	DIRECT_IO
CE2102U	RESET	IN_FILE	DIRECT_IO
CE2102V	OPEN	OUT_FILE	DIRECT_IO
CE2102W	RESET	OUT_FILE	DIRECT_IO
CE3102E	CREATE	IN_FILE	TEXT_IO
CE3102F	RESET	Any Mode	TEXT_IO
CE3102G	DELETE	———	TEXT_IO
CE3102I	CREATE	OUT_FILE	TEXT_IO
CE3102J	OPEN	IN_FILE	TEXT_IO
CE3102K	OPEN	OUT_FILE	TEXT_IO.

The following 16 tests check operations on sequential, direct, and text files when multiple internal files are associated with the same external file and one or more are open for writing; USE_ERROR is raised when this association is attempted.

CE2107B..E	CE2107G..H	CE2107L	CE2110B	CE2110D
CE2111D	CE2111H	CE3111B	CE3111D..E	CE3114B
CE3115A				

CE2203A checks that WRITE raises USE_ERROR if the capacity of an external sequential file is exceeded; this implementation cannot restrict file capacity.

CE2403A checks that WRITE raises USE_ERROR if the capacity of an external direct file is exceeded; this implementation cannot restrict file capacity.

CE3304A checks that SET_LINE_LENGTH and SET_PAGE_LENGTH raise USE_ERROR if they specify an inappropriate value for the external file; there are no inappropriate values for this implementation.

CE3413B checks that PAGE raises LAYOUT_ERROR when the value of the page number exceeds COUNT'LAST; for this implementation, the value of COUNT'LAST is greater than 150000, making the checking of this objective impractical.

2.3 TEST MODIFICATIONS

Modifications (see section 1.3) were required for 7 tests.

The following tests were split into two or more tests because this implementation did not report the violations of the Ada Standard in the way expected by the original tests.

B22003A   B83033B   B85013D

CA2009C and CA2009F were graded inapplicable by Evaluation Modification as directed by the AVO. These tests contain instantiations of a generic unit prior to the compilation of that unit's body; as allowed by AI-00408 and AI-00506, the compilation of the generic unit bodies makes the compilation unit that contains the instantiations obsolete.

BC3204C and BC3205D were graded passed by Processing Modification as directed by the AVO. These tests check that instantiations of generic units with unconstrained types as generic actual parameters are illegal if the generic bodies contain uses of the types that require a constraint. However, the generic bodies are compiled after the units that contain the instantiations, and this implementation creates a dependence of the instantiating units on the generic units as allowed by AI-00408 and AI-00506 such that the compilation of the generic bodies makes the instantiating units obsolete—no errors are detected. The processing of these tests was modified by re-compiling the obsolete units; all intended errors were then detected by the compiler.

CHAPTER 3

PROCESSING INFORMATION

3.1 TESTING ENVIRONMENT

The Ada implementation tested in this validation effort is described adequately by the information given in the initial pages of this report.

For technical and sales information about this Ada implementation, contact:

Michael LeBlanc
SKY Computers, Inc.
27 Industrial Ave.
Chelmsford MA 01824
(508) 250-1020

Testing of this Ada implementation was conducted at the customer's site by a validation team from the AVF.

3.2 SUMMARY OF TEST RESULTS

An Ada Implementation passes a given ACVC version if it processes each test of the customized test suite in accordance with the Ada Programming Language Standard, whether the test is applicable or inapplicable; otherwise, the Ada Implementation fails the ACVC [Pro92].

For all processed tests (inapplicable and applicable), a result was obtained that conforms to the Ada Programming Language Standard.

The list of items below gives the number of ACVC tests in various categories. All tests were processed, except those that were withdrawn because of test errors (item b; see section 2.1), those that require a floating-point precision that exceeds the implementation's maximum precision (item e; see section 2.2), and those that depend on the support of a file system — if none is supported (item d). All tests passed, except those that are listed in sections 2.1 and 2.2 (counted in items b and f, below).

PROCESSING INFORMATION

a) Total Number of Applicable Tests	3781	
b) Total Number of Withdrawn Tests	104	
c) Processed Inapplicable Tests	84	
d) Non-Processed I/O Tests	0	
e) Non-Processed Floating-Point Precision Tests	201	
f) Total Number of Inapplicable Tests	285	(c+d+e)
g) Total Number of Tests for ACVC 1.11	4170	(a+b+f)

3.3 TEST EXECUTION

A magnetic tape containing the customized test suite (see section 1.3) was taken on-site by the validation team for processing. The contents of the magnetic tape were loaded directly onto the host computer.

After the test files were loaded onto the host computer, the full set of tests was processed by the Ada implementation.

The tests were compiled and linked on the host computer system, as appropriate. The executable images were transferred to the target computer system by the Ethernet, and run. The results were captured on the host computer system.

Testing was performed using command scripts provided by the customer and reviewed by the validation team. See Appendix B for a complete listing of the processing options for this implementation. It also indicates the default options.

No explicit options were used for testing this implementation.

Test output, compiler and linker listings, and job logs were captured on magnetic tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

APPENDIX A

MACRO PARAMETERS

This appendix contains the macro parameters used for customizing the ACVC. The meaning and purpose of these parameters are explained in [UG89]. The parameter values are presented in two tables. The first table lists the values that are defined in terms of the maximum input-line length, which is the value for $MAX_IN_LEN--also listed here. These values are expressed here as Ada string aggregates, where "V" represents the maximum input-line length.

Macro Parameter	Macro Value
$MAX_IN_LEN	200 — Value of V
$BIG_ID1	(1..V-1 => 'A', V => '1')
$BIG_ID2	(1..V-1 => 'A', V => '2')
$BIG_ID3	(1..V/2 => 'A') & '3' & (1..V-1-V/2 => 'A')
$BIG_ID4	(1..V/2 => 'A') & '4' & (1..V-1-V/2 => 'A')
$BIG_INT_LIT	(1..V-3 => '0') & "298"
$BIG_REAL_LIT	(1..V-5 => '0') & "690.0"
$BIG_STRING1	'"' & (1..V/2 => 'A') & '"'
$BIG_STRING2	'"' & (1..V-1-V/2 => 'A') & '1' & '"'
$BLANKS	(1..V-20 => ' ')
$MAX_LEN_INT_BASED_LITERAL	"2:" & (1..V-5 => '0') & "11:"
$MAX_LEN_REAL_BASED_LITERAL	"16:" & (1..V-7 => '0') & "F.E:"

MACRO PARAMETERS

$MAX_STRING_LITERAL '"' & (1..V-2 => 'A') & '"'

The following table lists all of the other macro parameters and their respective values.

Macro Parameter	Macro Value
$ACC_SIZE	32
$ALIGNMENT	4
$COUNT_LAST	2_147_483_646
$DEFAULT_MEM_SIZE	1024
$DEFAULT_STOR_UNIT	8
$DEFAULT_SYS_NAME	BOLT
$DELTA_DOC	2.0**(-31)
$ENTRY_ADDRESS	16#0#
$ENTRY_ADDRESS1	16#1#
$ENTRY_ADDRESS2	16#2#
$FIELD_LAST	2_147_483_647
$FILE_TERMINATOR	' '
$FIXED_NAME	NO_SUCH_FIXED_TYPE
$FLOAT_NAME	NO_SUCH_FLOAT_TYPE
$FORM_STRING	""
$FORM_STRING2	"CANNOT_RESTRICT_FILE_CAPACITY"
$GREATER_THAN_DURATION	90_000.0
$GREATER_THAN_DURATION_BASE_LAST	10_000_000.0
$GREATER_THAN_FLOAT_BASE_LAST	3.5E+38
$GREATER_THAN_FLOAT_SAFE_LARGE	3.4E+38

$GREATER_THAN_SHORT_FLOAT_SAFE_LARGE	3.4E+38
$HIGH_PRIORITY	20
$ILLEGAL_EXTERNAL_FILE_NAME1	/NODIRECTORY/FILENAME1
$ILLEGAL_EXTERNAL_FILE_NAME2	/NODIRECTORY/FILENAME2
$INAPPROPRIATE_LINE_LENGTH	-1
$INAPPROPRIATE_PAGE_LENGTH	-1
$INCLUDE_PRAGMA1	PRAGMA INCLUDE ("A28006D1.ADA")
$INCLUDE_PRAGMA2	PRAGMA INCLUDE ("B28006F1.ADA")
$INTEGER_FIRST	-2147483648
$INTEGER_LAST	2147483647
$INTEGER_LAST_PLUS_1	2147483648
$INTERFACE_LANGUAGE	C
$LESS_THAN_DURATION	-90_000.0
$LESS_THAN_DURATION_BASE_FIRST	-10_000_000.0
$LINE_TERMINATOR	ASCII.LF
$LOW_PRIORITY	1
$MACHINE_CODE_STATEMENT	NULL;
$MACHINE_CODE_TYPE	INSTRUCTION
$MANTISSA_DOC	31
$MAX_DIGITS	15
$MAX_INT	2147483647
$MAX_INT_PLUS_1	2147483648
$MIN_INT	-2147483648
$NAME	BYTE_INTEGER

MACRO PARAMETERS

$NAME_LIST	BOLT
$NAME_SPECIFICATION1	/acvc/val/X2120A
$NAME_SPECIFICATION2	/acvc/val/X2120B
$NAME_SPECIFICATION3	/acvc/val/X3119A
$NEG_BASED_INT	16#FFFFFFFE#
$NEW_MEM_SIZE	1024
$NEW_STOR_UNIT	8
$NEW_SYS_NAME	BOLT
$PAGE_TERMINATOR	ASCII.LF & ASCII.FF
$RECORD_DEFINITION	NEW INTEGER
$RECORD_NAME	INSTRUCTION
$TASK_SIZE	32
$TASK_STORAGE_SIZE	2048
$TICK	1.0
$VARIABLE_ADDRESS	FCNDECL.VAR_ADDRESS
$VARIABLE_ADDRESS1	FCNDECL.VAR_ADDRESS1
$VARIABLE_ADDRESS2	FCNDECL.VAR_ADDRESS2
$YOUR_PRAGMA	NO_SUCH_PRAGMA

APPENDIX B

COMPILATION SYSTEM OPTIONS

The compiler options of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this appendix are to compiler documentation and not to this report.

ada

Invocation `ada [options . . .] file.ada . . .`

Description

The `ada` command invokes the SKYvec Ada compiler.

A program library must be created using `mklib` or `newlib` in advance of any compilation. The compiler aborts if it is unable to find a program library (either the default, ada.lib, in the current working directory or the library name specified with the `-L` option).

Note that the source file has the extension `.ada`. Just about any non-empty file extension is permitted. The ones not allowed include those used by the SKYvec Ada compiling system for other purposes such as `.o` for object module files. If an illegal extension is given, the error message "`missing or improper file name`" is displayed. Some other commonly used source file extensions are:

`.ads`	for package specification source files
`.adb`	for package body source files
`.sub`	for subunit (separate) source files

Options

`-fD` Generate debugging output. The `-fD` option causes the compiler to generate the appropriate code and data for operation with the SKYvec Ada Debugger. For more information on using this option and using the Debugger, see Chapter 9.

`-fE` Generate error log file. The `-fE` option causes the compiler to generate a log file containing all the error messages and warning messages produced during compilation. The error log file has the same name as the source file, with the extension `.err`. For example, the error log file for `simple.ada` is `simple.err`. The error log file is placed in the current working directory. In the absence of the `-fE` option, the error log information is sent to the standard output stream.

`-fL` Generate exception location information. The `-fL` option causes location information (source file names and line numbers) to be maintained for internal checks. This information is useful for debugging in the event that an "Exception never handled" message appears when an exception propagates out of the main program (see section *Exception Never Handled* on page 3-10). This option causes the code to be somewhat larger. If `-fL` is not used, exceptions that propagate out of the main program will behave in the same way, but no location information will be printed with the "Exception never handled" message.

`-fN` Suppress numeric checking. The `-fN` option suppresses two kinds of numeric checks for the entire compilation:

1. `division_check`
2. `overflow_check`

These checks are described in section 11.7 of the LRM. Using `-fN` reduces the size of the code. Note that there is a related `ada` option, `-fs` to suppress all checks for a compilation. See also section *Exception Never Handled* on page 3-10.

The `-fN` option must be used in place of pragma `suppress` for the two numeric checks, because presently pragma `suppress` is not supported for `division_check` and `overflow_check`. Pragma `suppress` works for other checks, as described in section *Reducing Program Size* on page 2-4. In the absence of the `-fN` option, the numeric checks are always performed.

`-fs` Suppress all checks. The `-fs` option suppresses all automatic checking, including numeric checking. This option is equivalent to using pragma `suppress` on all checks. This option reduces the size of the code, and is good for producing "production quality" code or for benchmarking the compiler. Note that there is a related `ada` option, `-fN` to suppress only certain kinds of numeric checks. See also sections *Reducing Program Size* on page 2-4 and *Automatic Checks* on page 3-11.

`-fv` Compile verbosely. The compiler prints the name of each subprogram, package, or generic as it is compiled. Information about the symbol table space remaining following compilation of the named entity is also printed in the form "[nK]".

`-fw` Suppress warning messages. With this option, the compiler does not print warning messages about ignored pragmas, exceptions that are certain to be raised at run-time, or other potential problems that the compiler is otherwise forbidden to deem as errors by the LRM.

`-g` The `-g` option instructs the compiler to run an additional optimization pass. The optimizer removes common sub-expressions, dead code and unnecessary jumps. It also does loop optimizations. This option is different from the `-g` option to `bamp`. The `-g` option to `ada` optimizes the specified unit when it is compiled; no inter-unit optimization is done. The `-g` option to `bamp` analyzes and optimizes the entire program at link time. Note: Even if `-g` is specified for the ada command, the `-K` option to `ada` must still be specified for the `-g` option to bamp to be effective.

`-K` Keep internal form file. This option is used in conjunction with the Optimizer (see Chapter for more information). Without this option, the compiler deletes internal form files following code generation.

`-l`*`modifiers`*

Generate listing file. The `-l` option causes the compiler to create a listing. Optional modifiers can be given to affect the listing format. You can use none or any combination of the following modifiers:

c continuous listing format

p	obey pragma `page` directives
s	use standard output
t	relevant text output only

The formats of and options for listings are discussed in section *Listings* on page 16-6. The default listing file generated has the same name as the source file, with the extension `.lst`. For example, the default listing file produced for simple.ada has the name `simple.lst`. The listing file is placed in the current working directory. Note: `-l` also causes an error log file to be produced, as with the `-fE` option.

`-L` *`library-name`*

Default: `ada.lib`

Use alternate library. `-L` option specifies an alternative name for the program library.

Note: Options beginning with `-f` can be combined, as in "`-fsv`." This is equivalent to specifying the options separately, e.g. "`-fs -fv`." Options beginning with `-l` can be similarly combined or separated, as in "`-lcs`" or "`-lc -ls`" (see section *Listings* on page 16-6).

Compiler Output Files

Files produced by compilations, other than listings and error logs, are:

Files	Description
`.atr`	interface description files
`.int`	Meridian Internal Form files
`.g`*`nn`*	generic description files; *nn* is a two-digit number
`.o`	object code files
`.sep`	subunit environment description files

Also produced are various intermediate files; these are usually deleted as a matter of course. You normally need not concern yourself with most of these output files with the exception of assembly language files.

Output files are placed either in the current working directory or in the auxiliary directory, depending on the configuration of the program library (as determined by `mklib` or `newlib`). The name of an auxiliary directory associated with a program library can be determined by using the `-h` option to the `lslib` command.

The name of an output file is derived from the first 10 characters of the compilation unit name, but when a name collision occurs, the library system assigns an arbitrary unique name that may bear no relation to the source file name (it might look like "aaaaaaab"). The `-l` option to the `lslib` command must be used to determine the base name used to derive the output file names for a particular library entry. The base name is displayed as the "Host system file name".

The name of an output file is derived from the compilation unit name.

Supplementary files that may be produced by a compilation are:

`.err`	error log files (only when `-fE` option used)
`.lst`	listing files (only when `-l` option used)

These are discussed in section *Listings* on page 16-6.

Non-Local Compilations

The compiler is able to compile files that reside in directories other than the current working directory. As always, a program library (typically `ada.lib`) must be present in the current working directory. All output files are placed in the current working directory or in the local auxiliary directory (`ada.aux`).

Compile-Time Error Messages

When syntactic or semantic errors are detected in the source code, the SKYvec Ada compiler produces either error messages or warning messages. These messages are normally produced on the standard output stream. If the `-fE` option is given, these messages are written, instead, to an error log file. The error log file has the same name as the source file, with the extension `.err`.

When error messages are printed, processing does not proceed beyond the first pass. No object code file is produced. Warning messages do not prevent further processing. Other passes (e.g. the code generator) may print error messages as well, but these are almost certain to be error messages related to problems internal to the compiler itself, and should be reported to SKY Computers, Inc.

Error messages have the form:

```
"filename", nn: English explanation of error [LRM l.m.n/p]
```

where *filename* is the name of the program source file in which the error was detected, *nn* is the specific line number in the source file where the error occurred, followed by an explanation in English of the error, and, when appropriate, a reference to the LRM. The LRM reference gives the chapter (*l*), section (*m*), subsection (*n*), and paragraph number (*p*).

An example follows.

```
"rt.ada", 245: record component redeclared [LRM 3.7/3]
```

Depending on the severity of the error, the SKYvec Ada compiler may attempt to recover and continue compiling the source code, or may terminate compilation immediately.

Warning messages have the form:

```
"filename", n: <<warning>> message
```

An example warning message is shown below.

```
"rt.ada", 297: <<warning>> INLINE: pragma has no effect
```

Error messages that begin with

```
*** Compiler Error
```

are internal compiler error messages. If any appear, they should be reported to SKY Computers.

All error messages and warning messages should be self-explanatory.

Listings

The compiler by default does not produce listings. The `-l` option causes the compiler to produce both a listing file and an error log file. The `-fE` option causes the compiler to produce only an error log file. In the absence of these options, the compiler prints an error log to the standard output stream alone.

Listing File Contents

The listing file contains line-numbered, paginated source text with error and warning messages interspersed. Listing file error messages appear in the format:

```
******E error message    *
```

Warning messages appear in the format:

```
++++++W warning message    +
```

The listing generated when `-l` is specified obeys pragma `list` and pragma `page` as described in the LRM. Pragma `list` and pragma `page` have no effect in the absence of the `-l` option.

Listing Format Control

Listing format is controlled via modifiers to the `-l` option or via a compiler default option description file named `ada.ini`. Refer to section *Default Option Description File* on page 16-7 for information about the `ada.ini` file.

The `-l` option has the form *`-lmodifiers`*, where *`modifiers`* are zero or more of these letters:

c Use continuous listing format. The listing by default contains a header on each page. Specifying `-lc` suppresses both pagination and header output, producing a continuous listing.

p Obey pragma `page` directives. Specifying `-lp` is only meaningful if `-lc` has also been given. Normally `-lc` suppresses all pagination, whereas `-lcp` suppresses all pagination except where explicitly called for within the source file with a pragma `page` directive.

s Use standard output. The listing by default is written to a file with the same name as the source file and the extension `.lst`, as in `simple.lst` from `simple.ada`. Specifying `-ls` causes the listing file to be written to the standard output stream instead. This output may be redirected anywhere (e.g. to the PRN device).

t Generate relevant text output only. The listing by default contains the entire source program as well as interspersed error messages and warning messages. Specifying `-lt` causes the compiler to list only the source lines to which error messages or warning messages apply, followed by the messages themselves.

Any, all or none of the suffix letters `c`, `p`, `s`, and `t` may be given following `-l`, as in `-lcs`, `-lct`, or `-lcst`. The options can also be given separately, as in `-lc -ls`.

Default Option Description File

Compiler behavior can be modified not only by command line options, but also by a default option description file named `ada.ini`. Default options are given in `lib/ada.ini` in the installation directory[1], while local overriding options can be given in an `ada.ini` file in the current working directory. At present, only listing format parameters can be set in a compiler default option description file.

The `ada.ini` file is an ordinary text file that may be created or edited with any editor used to edit Ada programs. The default `lib/ada.ini` file contents are:

```
--
-- SKYvec Ada compiler default option description file
--
--page_length          := 66;      -- min: 5
--top_margin           := 6;       -- min: 2
--bottom_margin        := 6;       -- min: 2
--left_margin          := 4;       -- min: 0
--page_width           := 132;     -- min: 40
--lineno_width         := 5;       -- min: 0
--marker_lines         := 1;       -- min: 0

--header_timestamp     := true
;--summary             := true
;--graphic_controls    := true;

--error_tag            := "E";--warning_tag:= "W";
```

The `lib/ada.ini` file as it is initially configured consists solely of comments showing examples of parameter assignments. The compiler default option description file may consist of Ada comments, blank lines, or assignments. In the example above, the initial comment characters ("- -") must be deleted to make any parameter assignment effective. The comments show the default values of the parameters, so there is no need to un-comment any particular assignment unless the value is to be changed. An example of a local `ada.ini` file might be:

```
---- Local SKYvec Ada compiler default option
-- description file
--
page_width :=  79;
-- Use all other defaults.
```

1. See your Software Release notes for installation information.

The parameters to which assignments can be made are:

`bottom_margin` This parameter sets the number of lines in the bottom margin of each page. The bottommost lines are blank. The minimum number of bottom margin lines that can be specified is 2. The bottom margin is suppressed by using the `-lc` (continuous listing) option.

`error_tag` This parameter specifies the character string that is displayed at the beginning of any error messages that occur in the listing. This string serves to highlight the error message.

`graphic_controls`
This parameter determines how non-printable characters in the Ada source file are printed. Non-printable characters include the ASCII DEL character and all ASCII control characters except for ASCII horizontal tab and the normal line terminator characters. If this parameter's value is true, then the control characters are printed as ^x where x is the printable character derived by adding the number 64 (decimal) to the ASCII code value of the control character. For example, the ASCII BEL character, also known as Control-G (^G), is printed as ^G. Other non-printable characters are displayed as ^?. If the parameter's value is set to false, then the non-printable characters are displayed as is with no conversion.

`header_timestamp`
This parameter specifies whether the heading information should include the date and time when the listing was generated. If its value is true, then the date and time are displayed. If its value is false, no date or time is printed.

`left_margin` This parameter specifies the number of blanks printed in each line before anything else (including line numbers, source lines, messages, and heading information). The left margin can be no smaller than 0 characters.

`lineno_width` This parameter specifies the number of characters reserved for the line number that appears to the left of each source line listed. The line number width can be no smaller than 0 characters. If 0 is specified, no line numbers are generated.

`marker_lines` This parameter specifies the number of marker lines to print before and after an error or warning message. Marker lines serve to make these messages stand out more from the rest of the listing. The minimum number of marker lines is 0.

`page_length` This parameter sets the number of lines printed per page. A page eject is placed at the end of each page. The minimum number of lines that can be specified for the page is 5. The

	page eject can be suppressed by using the `-lc` (continuous listing) option.
`page_width`	This parameter specifies the number of characters in the longest line that will be printed before the line is broken to the next line. The page width can be no smaller than 40 characters. This value does not include the number of characters specified by the `left_margin` and `lineno_width` parameters.
`summary`	This parameter specifies whether the listing should include a compilation summary at the end. If a summary is desired, this parameter's value should be set to true, otherwise it should be set to false.
`top_margin`	This parameter sets the number of lines in the top margin of each page. Centered in the topmost lines of each page a heading is printed containing a page number, file name, and date. The minimum number of top margin lines that can be specified is 2. The top margin and heading information can be suppressed altogether by using the `-lc` (continuous listing) option.
`warning_tag`	This parameter specifies the character string that is displayed at the beginning of any warning messages that occur in the listing. This string serves to highlight the warning message.

Examples

Example 34

Compile `x.ada` in the usual manner:

```
ada x.ada
```

Do not forget to type the extension. If you do not type the extension the `ada` program displays the error message "`missing or improper file name`".

Example 35

Compile `x.ada` with exception location maintenance code:

```
ada -fL x.ada
```

This is most useful when debugging a program that raises an exception.

Example 36

Compile `x.ada`, but with all automatic checking suppressed:

```
ada -fs x.ada
```

This is most useful after a program has been debugged and it is time to generate a "production" version of the program that is smaller and runs more quickly.

Example 37

Compile `x.ada`, but with numeric checking suppressed:

```
ada -fN x.ada
```

This retains most of the useful checks, while speeding up the program and decreasing its size.

Example 38

Compile `x.ada`, `y.ads`, and `z.adb` in the usual manner:

```
ada x.ada y.ads z.adb
```

Each source file is compiled in the order given.

Example 39

Compile `x.ada` using an alternate program library named `z.lib`:

```
ada -L z.lib x.ada
```

This presumes that `z.lib` was created in the current working directory with the mklib program prior to compilation.

Example 40

Compile `x.ada` verbosely, suppressing warning messages:

```
ada -fvw x.ada
```

For the `-fv` option, the compiler prints the name of each subprogram, package, or generic as it is compiled, along with the amount of symbol table space remaining. For the `-fw` option, the compiler suppresses warning messages.

Example 41

Compile `simple.ada`, producing a listing file:

```
ada -l simple.ada
```

This produces a listing file named `simple.lst` and an error log file, `simple.err`.

Example 42

Compile `simple.ada`, producing a continuous-form listing file on standard output:

```
ada -lsc simple.ada
```

This produces a listing on standard output in continuous format (no headers or pagination), as well as an error log file, `simple.err`.

Example 43

Compile `simple.ada` in a non-local directory, producing a local object file:

```
ada /x/ada/simple.ada
```

An `ada.lib` file must be present in the current directory. All output files are placed in the current directory or in the local auxiliary directory (`ada.aux`).

Example 44

Compile simple.ada normally, but retain information for the global optimizer:

```
ada -K simple.ada
```

Runs the compiler normally, but does not delete `simple.int`, which can be used by a subsequent global optimization (`bamp -g`) command.

LINKER OPTIONS

The linker options of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this appendix are to linker documentation and not to this report.

bamp

Invocation

```
bamp [options...] [main-procedure-name]
```

Description

The bamp (Build Ada Main Program) command creates an executable program given (an MPW tool by default, see section) the name of the main subprogram. The main-procedure-name given to bamp must be a parameterless procedure that has already been compiled.

Note: Be careful not to confuse the name of the source file containing the main subprogram (e.g. simple.ada) with the actual name of the main subprogram (e.g. simple).

If a main-procedure-name is not specified on the bamp command line, bamp links using the last-compiled subprogram that fits the profile for a main subprogram. To determine which subprogram will be used when no main subprogram is given to bamp, use the lslib -t option. When in doubt, it may be best to specify the main subprogram explicitly.

Note that when no main subprogram is specified, bamp selects the most recently compiled subprogram, not the most recently linked subprogram. If several different main subprograms are linked between compiles, still the most recently compiled subprogram is selected if no subprogram is explicitly specified.

The bamp program functions as a high-level linker. It works by creating a top-level main program that contains all necessary context clauses and calls to package elaboration procedures. The main program is created as an internal form file on which the code generator is run. Following this code generation pass, all the required object files are linked.

An optional optimization pass can be invoked via the bamp command. The details of optimization are discussed in Chapter 7. The bamp options relevant to optimization, -g and -G , are discussed below.

Programs compiled in Debug mode (with the xada -fD option) are automatically linked with the SKYvec Ada source level debugger.

Options

-A Aggressively inline. This option instructs the optimizer to aggressively inline subprograms when used in addition to the -G option. Typically, this means that subprograms that are only called once are inlined. If only the -G option is used, only subprograms for which pragma inline has been specified are inlined.

-c compiler-program-name
Default: As stored in program library. Use alternate compiler. Specifies the complete (nonrelative) directory path to the SKYvec Ada compiler. This option overrides the compiler program name stored in the program library. The -c option is intended for use in cross-compiler configurations, although under such circumstances, an appropriate library configuration is

normally used instead.

-f Suppress main program generation step. Suppresses the creation and additional code generation steps for the temporary main program file. The -f option can be used when a simple change has been made to the body of a compilation unit. If unit elaboration order is changed, or if the specification of a unit is changed, or if new units are added, then this option should not be used. The -f option saves a few seconds, but places an additional bookkeeping burden on you. The option should be avoided under most circumstances. Note that invoking bamp with the -n option followed by another invocation of bamp with the -f option has the same effect as an invocation of bamp with neither option (-n and -f neutralize each other).

-g Perform global optimization only. Causes bamp to invoke the global optimizer on your program. Compilation units to be optimized globally must have been compiled with the xada -K option.

-G Causes bamp to perform both global and local optimization. This includes performing pragma inline. As with the -g option, compilation units to be optimized must have been compiled with the xada -K option.

-I Link the program with a version of the tasking run-time which supports pre-emptive task scheduling. Produces code which handles interrupts more quickly, but has a slight negative impact on performance in general.

-L library-name
Default: ada.lib. Use alternate library, specifies the name of the program library the bamp program consults. (Overrides default library name).

-n No link. Suppresses actual object file linkage, but creates and performs code generation on the main program file. Note that invoking bamp with the -n option followed by another invocation of bamp with the -f option has the same effect as an invocation of bamp with neither option. That is, -n and -f neutralize each other.

-N No operations. Causes the bamp command to do a "dry run"; it prints out the actions it takes to generate the executable program, but does not actually perform those actions. Similar to the -P option.

-o output-file-name
Default: file. Use alternate executable file output name. Specifies the name of the executable program file written by the bamp command. This option overrides the default output file name.

cont.bamp

-P Print operations. Causes the bamp command to print out the actions it takes to generate the executable program as the actions are performed.

-v Link verbosely. Causes the bamp command to print out information about what actions it takes in building the main program such as:

- The name of the program library consulted.

- The library search order (listed as "saves" of the library units used by the program).
- The name of the main program file created (as opposed to the main procedure name).
- The elaboration order.
- The total program stack size.
- The name of the executable load module created.
- The verbose code generation for the main program file.

-W Suppress warnings. Allows you to suppress warnings from the optimizer.

APPENDIX C

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in Chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this Appendix are to compiler documentation and not to this report. Implementation-specific portions of the package STANDARD, which are not a part of Appendix F, are:

```
package STANDARD is
   ..........
   type INTEGER is range -2147483648 .. 2147483647;
   type LONG_INTEGER is range -2147483648 .. 2147483647;
   type SHORT_INTEGER is range -32768 .. 32767;
   type BYTE_INTEGER is range -128 ..127;

   type FLOAT is digits 6 range -3.40282E+038 .. 3.40282E+038;
   type LONG_FLOAT is digits 15
                range -1.79769313486231E+308 .. 1.79769313486231E+308;

   type DURATION is delta 0.0001 range -86400.0 .. 86400.0;
   ..........
end STANDARD;
```

Appendix F Implementation-Dependent Characteristics

This appendix lists implementation-dependent characteristics of SKYvec Ada. Note that there are no preceding appendices. This appendix is called Appendix F in order to comply with the *Reference Manual for the Ada Programming Language* (LRM) ANSI/MIL-STD-1815A which states that this appendix be named Appendix F.

Implemented Chapter 13 features include length clauses, enumeration representation clauses, address clauses, interrupts, package `system`, machine code insertions, pragma `interface`, and unchecked programming.

Pragmas

The implemented pre-defined pragmas[1] are:

`elaborate`	See the LRM section 10.5.
`interface`	See section *Pragma Interface* on page F-2.
`list`	See the LRM Appendix B.
`pack`	See section *Pragma Pack* on page F-3.
`page`	See the LRM Appendix B.
`priority`	See the LRM Appendix B.
`suppress`	See section *Pragma Suppress* on page F-4.
`inline`	See the LRM section 6.3.2[2].

The remaining predefined pragmas are accepted, but presently ignored:

```
controlled          optimize            system_name
shared              storage_unit        memory_size
interface
```

1. packNamed parameter notation for pragmas is not supported.
2. This pragma is not actually effective unless you compile/link your program using the global optimizer.

When illegal parameter forms are encountered at compile time, the compiler issues a warning message rather than an error, as required by the Ada language definition. Refer to the LRM Appendix B for additional information about the pre-defined pragmas.

Pragma Interface

The form of pragma `interface` in SKYvec Ada is:

```
pragma interface( language, subprogram [, "link-name"]);
```

where:

`language`	This is the interface language, one of the names `assembly`, `builtin`, `c`, or `internal`. The names `builtin` and `internal` are reserved for use in run-time support packages.
`subprogram`	This is the name of a subprogram to which the pragma `interface` applies. If `link-name` is omitted, then the Ada subprogram name is also used as the object code symbol name. Depending on the language specified, some automatic modifications may be made to the object code symbol name.
`link-name`	This is an optional string literal specifying the name of the non-Ada subprogram corresponding to the Ada subprogram named in the second parameter. If `link-name` is omitted, then `link-name` defaults to the value of subprogram translated to lowercase. Depending on the language specified, some automatic modifications may be made to the link-name to produce the actual object code symbol name that is generated whenever references are made to the corresponding Ada subprogram. The object code symbol generated for `link-name` is always translated to upper case. Although the object linker is case-sensitive, it is a rare object module that contains mixed-case symbols.It is appropriate to use the optional `link-name` parameter to pragma `interface` only when the interface subprogram has a name that does not correspond at all to its Ada identifier or when the interface subprogram name cannot be given using rules for constructing Ada identifiers (e.g. if the name contains a '$' character).

The characteristics of object code symbols generated for each interface language are as follows:

`assembly`	The object code symbol is the same as `link-name`. If no `link-name` string is specified, then the subprogram name is translated to lowercase.

`builtin`	The object code symbol is the same as `link-name`, but prefixed with one underscore character ("_"), whether or not a link-name string is specified two underscore characters ("__"). This language interface is reserved. The `builtin` interface is presently used to declare certain low-level run-time operations whose names must not conflict with programmer-defined or language system defined names.
`c`	The object code symbol is the same as `link-name`, but with one underscore character ('_') prepended. This is the convention used by the C compiler.subprogram name. If no `link-name` string is specified, then the subprogram name is translated to lowercase.
`internal`	No object code symbol is generated for an internal language interface; this language interface is reserved. The internal interface is presently used to declare certain machine-level bit operations.

No automatic data conversions are performed on parameters of interface subprograms. It is up to the programmer to ensure that calling conventions match and that any necessary data conversions take place when calling interface subprograms.

A pragma `interface` may appear within the same declarative part as the subprogram to which the pragma `interface` applies, following the subprogram declaration, and prior to the first use of the subprogram. A pragma `interface` that applies to a subprogram declared in a package specification must occur within the same package specification as the subprogram declaration; the pragma `interface` may not appear in the package body in this case. A pragma `interface` declaration for either a private or nonprivate subprogram declaration may appear in the private part of a package specification.

Pragma `interface` for library units is not supported.

Refer to the LRM section 13.9 for additional information about pragma interface.

Pragma Pack

Pragma `pack` is implemented for composite types (records and arrays).

Pragma `pack` is permitted following the composite type declaration to which it applies, provided that the pragma occurs within the same declarative part as the composite type declaration, before any objects or components of the composite type are declared.

Note that the declarative part restriction means that the type declaration and accompanying pragma `pack` cannot be split across a package specification and body.

The effect of pragma `pack` is to minimize storage consumption by discrete component types whose ranges permit packing. Use of pragma `pack` does not defeat allocations of alignment storage gaps for some record types. Pragma `pack` does not affect the representations of real types, pre-defined integer types, and access types.

Pragma Suppress

Pragma suppress is implemented as described in the LRM section 11.7, with these differences:

- Presently, division_check and overflow_check must be suppressed via a compiler flag, -fN; pragma suppress is ignored for these two numeric checks.
- The optional "ON =>" parameter name notation for pragma suppress is ignored.
- The optional second parameter to pragma suppress is ignored; the pragma always applies to the entire scope in which it appears.

Attributes

All attributes described in the LRM Appendix A are supported.

Standard Types

Additional standard types are defined in SKYvec Ada:

- byte_integer
- short_integer
- long_integer

The standard numeric types are defined as:

Type	Range
byte_integer	-128 .. 127;
short_integer	-32768 .. 32767;
integer	-2147483648 .. 2147483647;
long_integer	-2147483648 .. 2147483647;
float (6 digits)	-3.40282E+038 .. 3.40282E+038;
long_float (15 digits)	-1.79769313486231E+308 .. 1.79769313486231E+308;
duration (0.0001 delta)	-86400.0000 .. 86400.0000;

Package System

The specification of package system is:

package system is

type address is:	new integer;
type name is:	(BOLT);
system_name:	constant name:= BOLT;

```
        storage_unit:        constant:= 8;
        memory_size:         constant:= 1024;

-- System-Dependent Named Numbers

        min_int:             constant:= -2147483648;
        max_int:             constant:= 2147483647;
        max_digits:          constant:= 15;
        max_mantissa:        constant:= 31;
        fine_delta:          constant:= 2.0 ** -31;
        tick:                constant:= 1.0;

-- Other System-Dependent Declarations

        subtype priority is integer range 1.. 20;
```

The value of `system.memory_size` is presently meaningless.

Restrictions on Representation Clauses

Length Clauses

A size specification (`t'size`) is rejected if fewer bits are specified than can accommodate the type. The minimum size of a composite type may be subject to application of pragma `pack`. It is permitted to specify precise sizes for unsigned integer ranges, e.g. 8 for the range 0..255. However, because of requirements imposed by the Ada language definition, a full 32-bit range of unsigned values, i.e. 0..(2^{32}-1), cannot be defined, even using a size specification.

The specification of collection size (`t'storage_size`) is evaluated at run-time when the scope of the type to which the length clause applies is entered, and is therefore subject to rejection (via `storage_error`) based on available storage at the time the allocation is made. A collection may include storage used for run-time administration of the collection, and therefore should not be expected to accommodate a specific number of objects. Furthermore, certain classes of objects such as unconstrained discriminant array components of records may be allocated outside a given collection, so a collection may accommodate more objects than might be expected.

The specification of storage for a task activation (`t'storage_size`) is evaluated at run-time when a task to which the length clause applies is activated, and is therefore subject to rejection (via `storage_error`) based on available storage at the time the allocation is made. Storage reserved for a task activation is separate from storage needed for any collections defined within a task body.

The specification of small for a fixed point type (`t'small`) is subject only to restrictions defined in the LRM section 13.2.

Enumeration Representation Clauses

The internal code for the literal of an enumeration type named in an enumeration representation clause must be in the range of `standard.integer`.

The value of an internal code may be obtained by applying an appropriate instantiation of `unchecked_conversion` to an `integer` type.

Record Representation Clauses

The storage unit offset (the `at static_simple_expression` part) is given in terms of 8-bit storage units and must be even.

A bit position (the range part) applied to a discrete type component may be in the range 0..15, with 0 being the least significant bit of a component. A range specification may not specify a size smaller than can accommodate the component. A range specification for a component not accommodating bit packing may have a higher upper bound as appropriate (e.g. 0..31 for a discriminant string component). Refer to the internal data representation of a given component in determining the component size and assigning offsets.

Components of discrete types for which bit positions are specified may not straddle 16-bit word boundaries.

The value of an alignment clause (the optional `at mod` part) must evaluate to 1, 2, 4, or 8, and may not be smaller than the highest alignment required by any component of the record.

Address Clauses

An address clause may be supplied for an object (whether constant or variable) or a task entry, but not for a subprogram, package, or task unit. The meaning of an address clause supplied for a task entry is given in section *Interrupts* on page F-7.

An address expression for an object is a 32-bit linear segmented memory address of type `system.address`.

Interrupts

A task entry's address clause can be used to associate the entry with a UNIX signal. Values in the range 0..31 are meaningful, and represent the signals corresponding to those values.

An interrupt entry may not have any parameters.

Change of Representation

There are no restrictions for changes of representation effected by means of type conversion.

Implementation-Dependent Components

No names are generated by the implementation to denote implementation-dependent components.

Unchecked Conversions

There are no restrictions on the use of `unchecked_conversion`. Conversions between objects whose sizes do not conform may result in storage areas with undefined values.

Input-Output Packages

A summary of the implementation-dependent input-output characteristics is:

- In calls to `open` and `create`, the form parameter must be the empty string (the default value).
- More than one internal file can be associated with a single external file for reading only. For writing, only one internal file may be associated with an external file. Do not use reset to get around this rule.
- Temporary sequential and direct files are given names. Temporary files are deleted when they are closed.
- File I/O is buffered; text files associated with terminal devices are line-buffered.
- The packages `sequential_io` and `direct_io` cannot be instantiated with unconstrained composite types or record types with discriminants without defaults.

Source Line and Identifier Lengths

Source lines and identifiers in Ada source programs are presently limited to 200 characters in length.